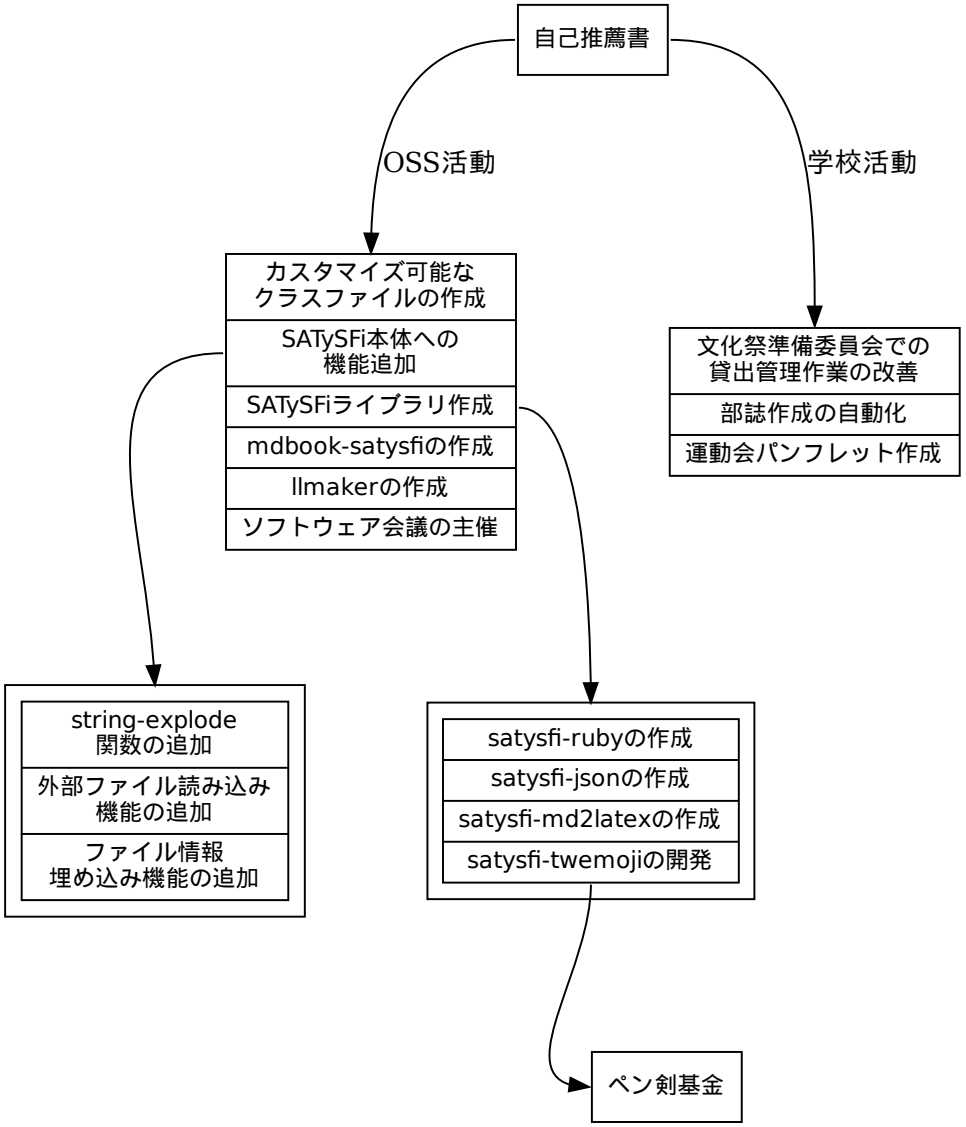
はじめに

筑波大学 AC 入試で使った自己推薦書用クラスファイルのデモファイルです。

自己推薦書の本文の一部を残しています。

ちなみに、下のチャート図の通りにはなっていません。公開したくない箇所は削除しているので。

この自己推薦書の章立ては以下の図のような構成となっている。



目次

1. OSS 活動

1.1. SATySFI でのカスタマイズが容易なクラスファイルの作成について

中三学年の時に行われた「Y2b」という論文作成課題の際、「SATySFI でのカスタマイズが容易なクラスファイルの作成」というテーマで研究を行い、発表を行った。

SATySFI（サティスファイ）は「LaTeX よりも良い組版ソフトウェアを作る」を目標に作成されている組版ソフトウェアで、2017 年度 IPA 未踏事業により開発された。現在も GitHub 上で開発が続けられている。発音は英単語の"satisfy"と同じである。

この SATySFI というソフトウェアで文書を作る際には、LaTeX と同じように「クラスファイル」と呼ばれる、「文章構造や、フォント・デザイン・版面設計などのレイアウトに関する設定をするライブラリ」が必要となる。

LaTeX のクラスファイルでは設定を豊富に行うことができ、さらに各種パッケージを使うことでレイアウトに関する高度な変更ができるようになっている。しかし SATySFI で当時提供されていたクラスファイルはそのような設定をすることができず、用紙サイズを A4 から B5 に変更したいだけでクラスファイルを一から作らなければならないような状況であった。

そこで、「文書を書く人が文書の用途によって容易に版面などのデザインを変更できるようにする」クラスファイルを作成しようと考えた。

まず、LaTeX のクラスファイルでどのようなオプションがあるかをドキュメントを読んで調査した。また、LaTeX でデザインやレイアウトを変更できるパッケージを検索し、どのような部分を変更する需要があるのかについて調査した。

SATySFI では特別な設定をしなければ値の上書きをできない。また、そもそもとしてレイアウトやデザインに関わる部分はクラスファイルが責任をもって管理するべき部分である。そのため、LaTeX のような外部パッケージによる上書きでのレイアウトの変更は採用せず、クラスファイルで全て操作できるようにした。変更したいパラメータについて案を固めた後に、パラメータに値を渡す方法について考えた。その結果、

- 既存のクラスファイルを踏襲し、文書を作る main 関数にパラメータを引数の形で渡して実現する
- わかりやすいようにスタイル・デザイン・フォントなどにパラメータの要素を分割する
- パラメータに与える値を定数としていくつか用意しておき、用途に応じて使いまわせるようにする

という仕様に落ち着いた。この結果、今まであったクラスファイルと似たような使い勝手な上に細かく制御ができ、さらに一度スタイルを決めたものを使いまわせるようになる、という非常に使いやすい形となった。

上記のように仕様が固まったのでこれを実装した。その際、既存のクラスファイルの実装を参考にしながら実装した。この時、パラメータの値を取り出すときに苦労をした。結局、事前にダミーの値を用意しておき、それを内部で上書きをするという方法をとった。

用紙サイズを A4 にするスタイルや B5 にするスタイル、レポート用のデザインや文書用のデザインなど、様々なスタイルをデフォルトで用意しておいたため、文書作成者が用途に応じて容易に版面やデザインを変更できるようになり、当初の目標である、「文書を書く人が文書の用途によって容易に版面などのデザインを変更できるようにするクラスファイルを作成する」という目標を達成した。

実装したものは"ExDesign"という名前で GitHub リポジトリ [2] で公開・配布している。

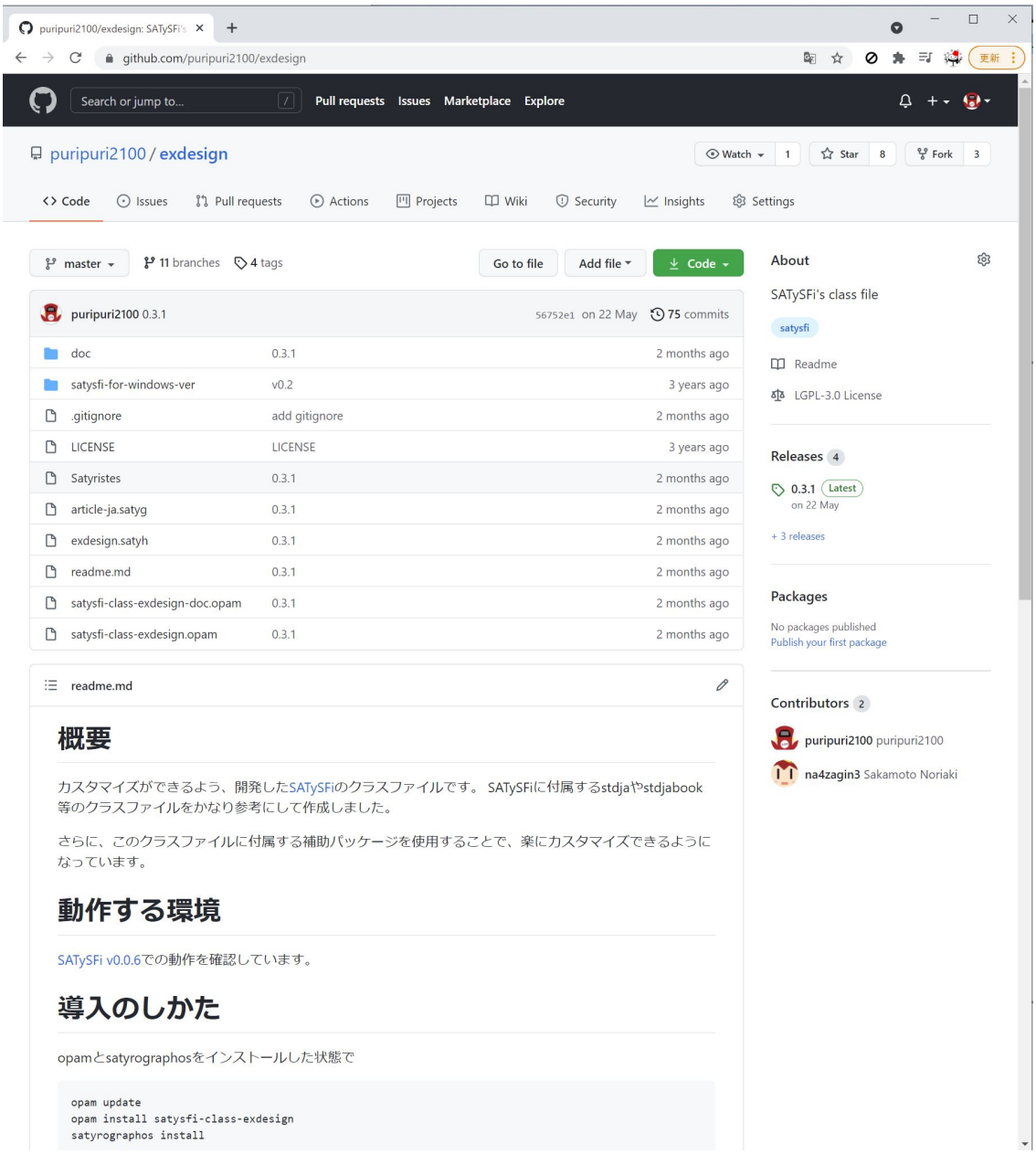


図 1 exdesign の GitHub リポジトリ [2]

1.2. SATγSFI 本体への機能追加

SATγSFI の拡張機能を作成するだけではなく、SATγSFI 本体に機能追加の提案をし、取り込まれたものがいくつかある。ここではそれを紹介する。

1.2.1. string-explode 関数の追加

SATγSFI では文字は「文字列」としてしか扱うことができず、文字単体を表す表現が無かった。また、文字の比較を行う機能も無かった。そのため、文字列を key とした sort を行うこともできず、文字列の評価もあまり上手くできていなかった。

これを解消するため、文字列を Unicode ポイント列に変換するプリミティブを実装し、それを追加する提案を行った。既に「SATγSFI での文字列を実装言語での文字単位ごとに分割する関数」と「文字単位を Unicode ポイントに変換する関数」が実装として用意されていたため、これを組み合わせることで実現することができた。

関数追加の提案は SATγSFI の GitHub リポジトリで行った。[3]

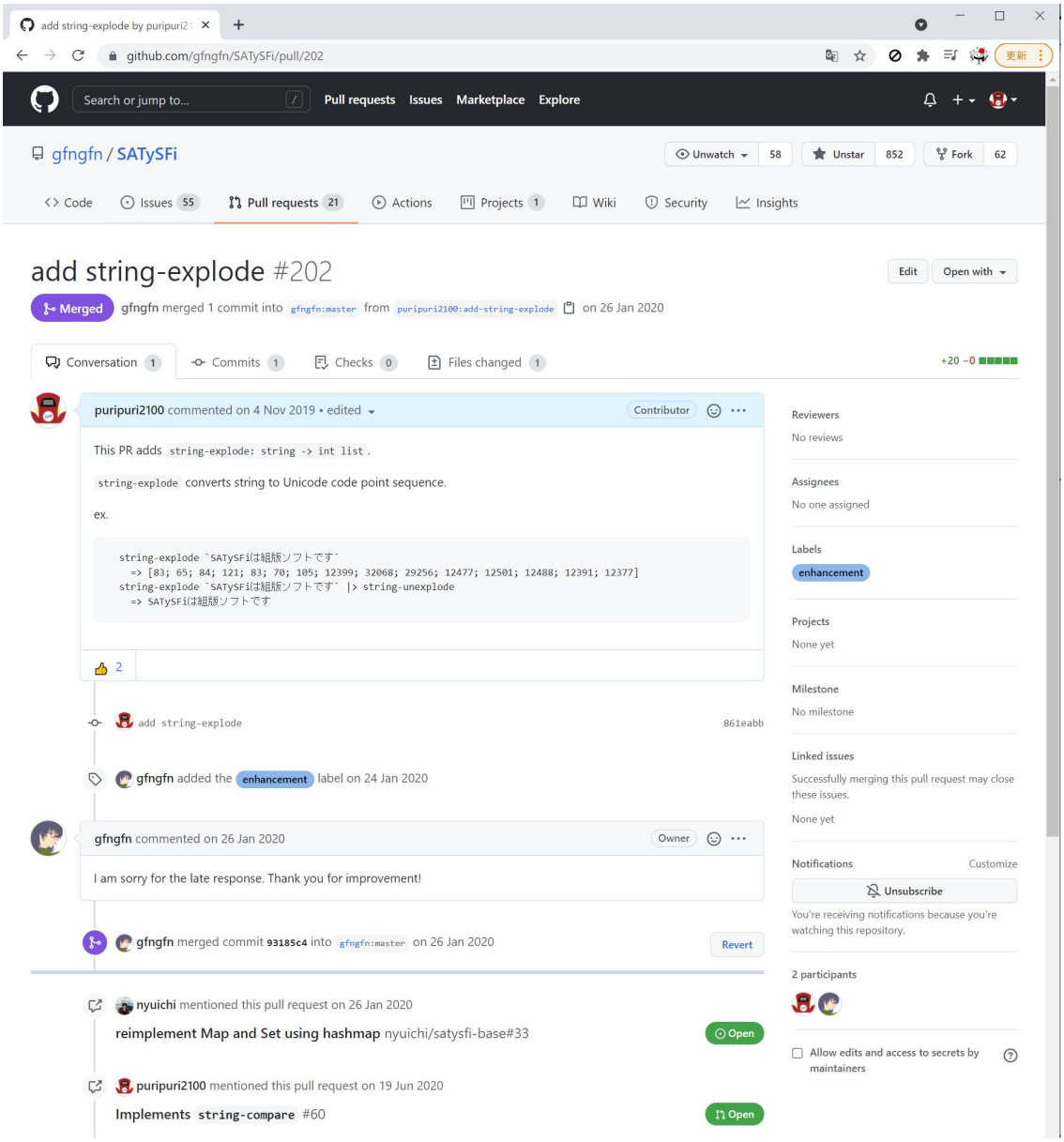


図 2 Pull Request #202[3]

このプリミティブが追加されたことで以下のような拡張機能を作成することができるようになった。

- 文字列の比較関数の実装 [4]
- 文字列から漢数字・ローマ数字への変換関数の実装 [5]

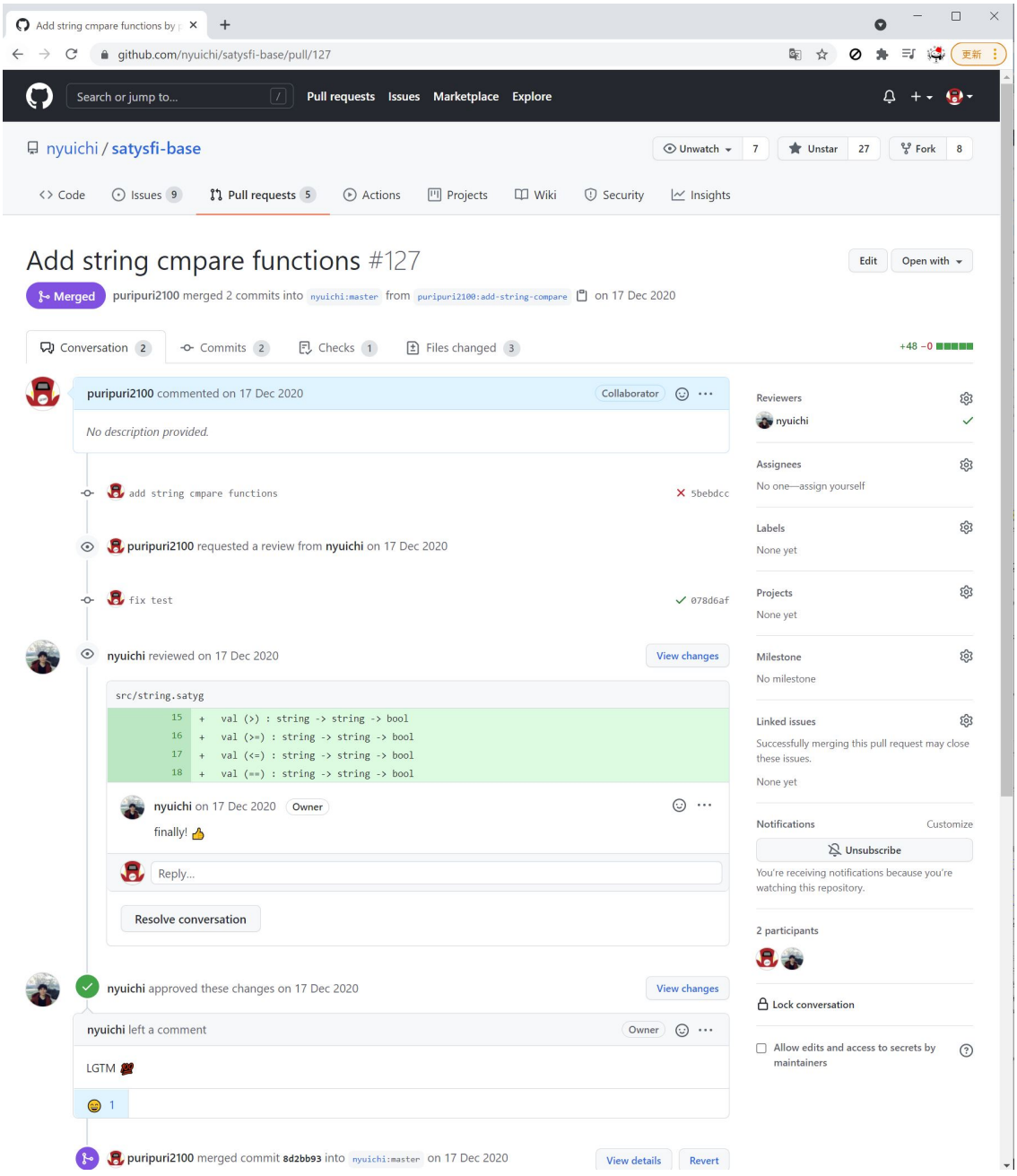


図 3 Pull Request #127[4]

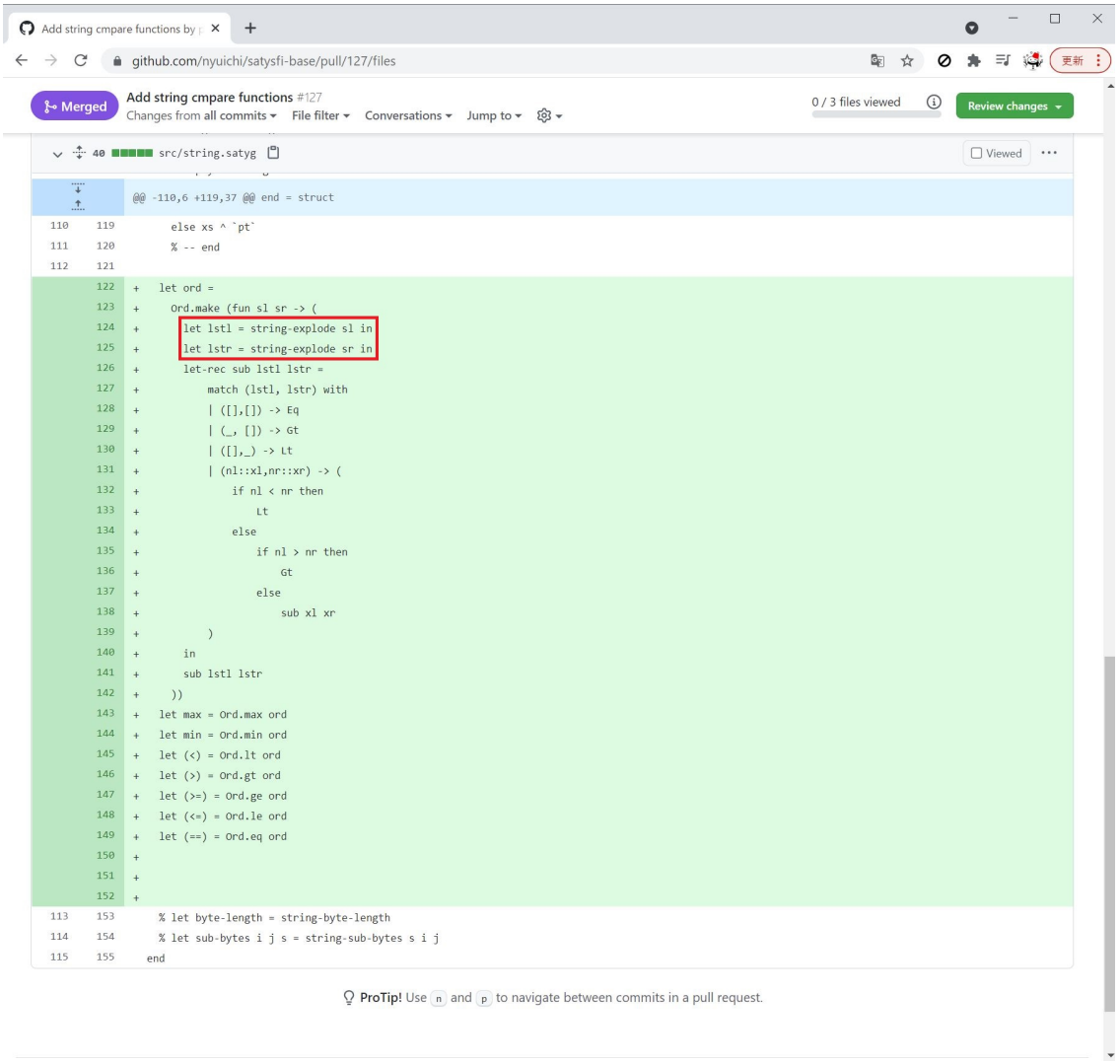


図 4 Pull Request #127 での string-explode の使い場所 [4]

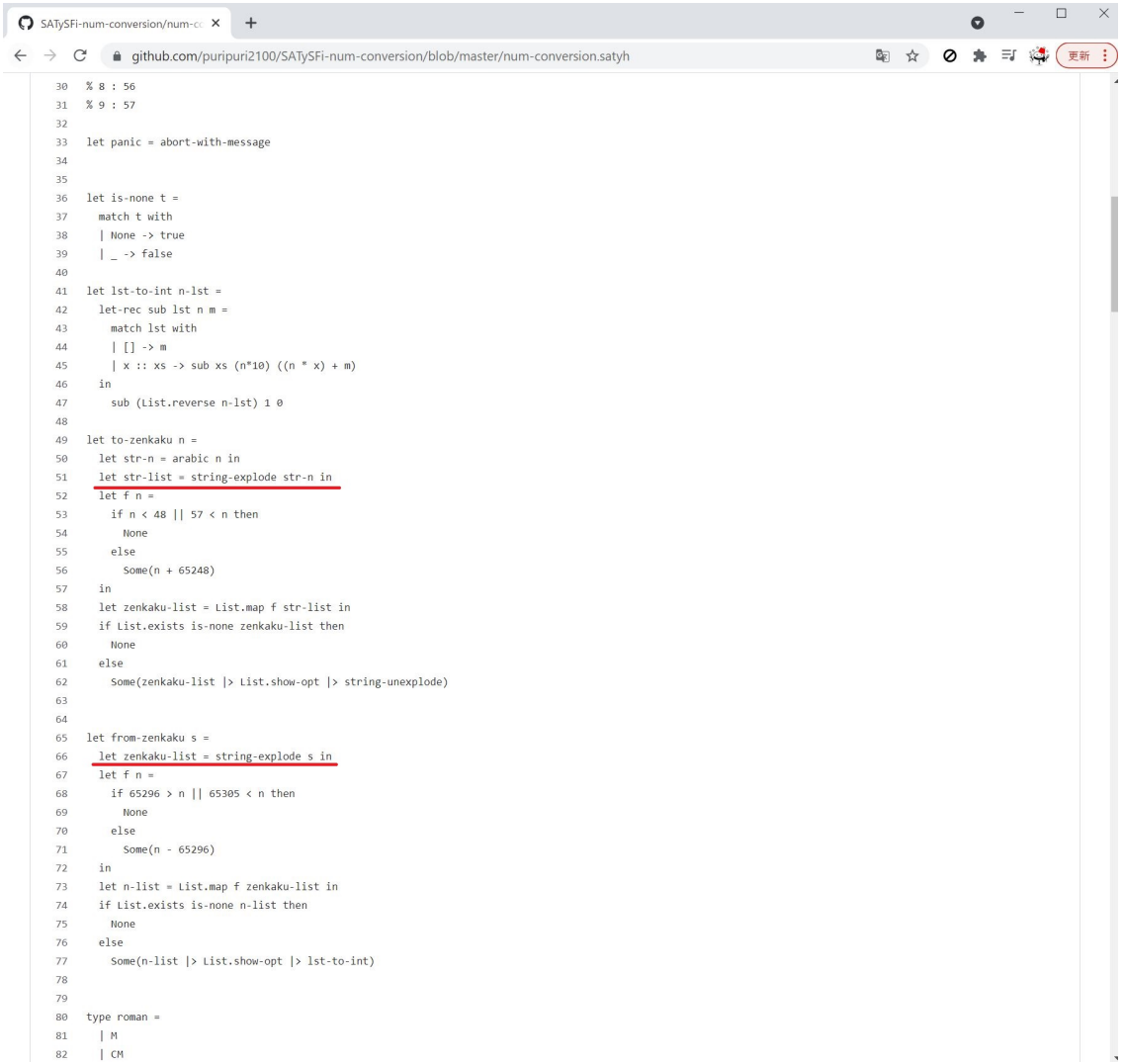


図 5 num-conversion での string-explode の使い場所 [5]

1.3. SATySFi のライブラリの作成

SATySFi は組版層とプログラミング層に機能が分かれている、どちらに対しても拡張機能をユーザーが用意することができる。私は今まで 25 種類以上もの拡張機能を作成し、ライブラリとして公開・配布している。その中でも特に記憶に強く残っているものを紹介する。

1.3.1. satysfi-ruby の作成

小説を作成する際にルビ（振り仮名）は必要となってくる。ルビは HTML の規格にも <ruby> タグとして実装されているほど国際的にメジャーなものになっている。SATySFi で小説を書こうと思った人にとって、ルビを振る機能が無いことがハードルにならないようにルビを振る機能を実装することにした。

ルビを振るコマンドの仕様を決める際には、日本語の組版処理の例示と解説を W3C が公開している「日本語組版処理の要件 (JLreq)」というものを参考にした。JLreq の中に

「ルビと圏点処理」という節が設けられており、そこに詳しくまとめられていた。[9]

JLreq によるとルビには、漢字一文字に対してその漢字の読み方を振る「モノルビ」と、熟字訓のように分割できない一単語に対してその読み方を振る 「グループルビ」、 そして熟語一つに対してその熟語の読み方を振る「熟語ルビ」の 3 種類があるとされている。

図 6 モノルビの例

図 7 グループルビの例

熟語ルビは「流儀(りゅうぎ)」のような、漢字ごとの読みの長さが極端に違う場合に、漢字間に余計なスペースが入らないようにルビを違う漢字にかかるように調節する振り方をするものである（この場合、モノルビであると「流(りゅう) 儀(ぎ)」となり、「流」と「儀」の間にスペースが入ってしまう。）。

図 8 左のモノルビと右の熟語ルビとの文字間スペースの比較

ルビを振るためのコマンドの入力は統一して「漢字一文字ずつのリスト」と「漢字一文字に対応する読みのリスト」の 2 つを引数とした。オプションでルビの種類を切り替えたり、左右のスペースを調節できるようにした。このインターフェースは LaTeX でルビ機能を提供するパッケージである"PXrubrica"パッケージのそれを参考にした。

モノルビは次のように実装した。

まず、2 つの引数を基にルビと漢字をペアにしたリストを作成する。次にルビ部分と漢字部分を自然に組む。そしてその組んだ結果の横の長さを計測する。そしてその長さをそれぞれ比較し、ルビ文字の方が短い場合には漢字一文字に対して中央揃えの位置にスペースを調節して出力する。 ルビ文字の方が長い場合はその逆にルビ文字に対して漢字の方を中央揃えとする。

グループルビは次のように実装した。

まず、モノルビと同じようにルビと漢字をペアにしたリストを作成する。そしてルビ部分と漢字部分を自然に組み、長さを計測する。その後ルビ部分が短い場合にはルビ同士の間にスペースを、漢字部分が短い場合には漢字同士の間にスペースを入れることでルビ部分と漢字部分の横の長さが揃うようにする。その際、見栄えを整えるために均一にスペースを割り振るのではなく、 両端のスペースの大きさが文字間のスペースの大きさの丁度半分になるように調節する。

JRLeq では熟語ルビの実装方法について、熟語の構成やその熟語の前後に来る文字で振り仮名の振り方やスペースを調節する方法と、「日本語文書の組版方法」という名称の JIS の規格である “JIS X 4051” に規定された方法の 2 通りが挙げられている。SATɣSFI では前後の文字を取得する方法がないため、 "JIS X 4051"で規定された方法で実装することとした。

"JIS X 4501"では「ルビが漢字よりも長い文字が一文字でもある」 場合と、そうでない場合とに分けて処理を行うことと規定されている。「ルビが漢字よりも長い文字が一文字でもある」 場合にはグループルビと同じように組み、 そうでない場合には熟語ルビと同じように組むように規定されている。 そのため、 それに従い、 ルビ部分と漢字部分を一旦組んでそれぞれの長さを計測し、 その長さを比較した後に、 モノルビとグループルビに処理を振り分けるように実装した。

各ルビの種類ごとに処理方法は以上のように実装したが、ルビ部分を漢字部分の上に素直に載せる処理を行うと、「一」などの高さがあまりない漢字の部分だけルビの位置が下がってしまい、見栄えがとても悪くなる。そのため、漢字部分を組んだ後に全ての漢字について高さの計測を行い、一番高いところにルビの高さが揃うようにブロックを挿入するという調節処理を行っている。

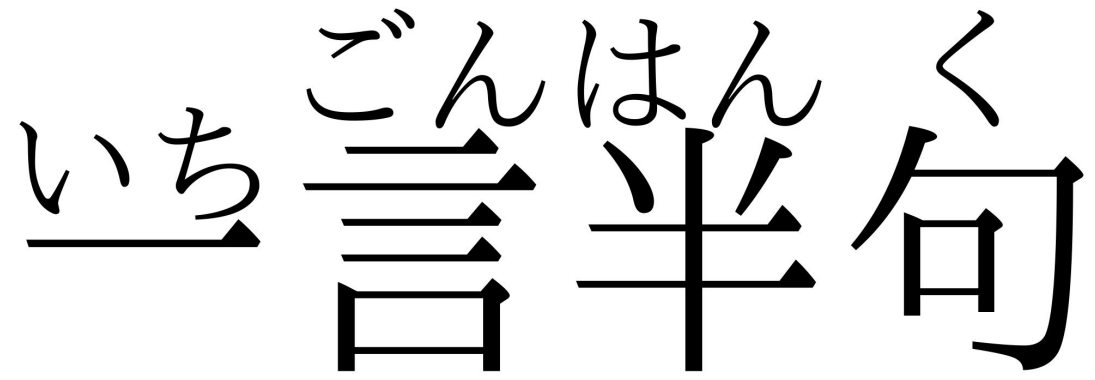


図 9 高さの揃わないルビの例

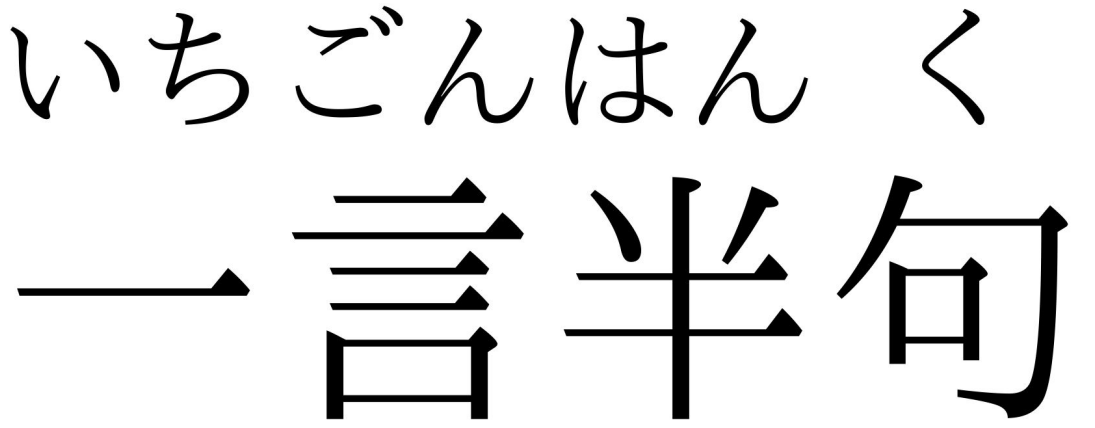


図 10 高さを調節した後のルビの例

このライブラリにはルビを組む以外にも調節機能が備わっている。例えば、ルビ部分が漢字部分よりも大きくはみ出している場合には前後の文字にルビ部分をはみ出させることができると JLReq には書かれている。SATYSFI では前後の文字を取得することができないため、ここは使用者がオプションで手動で行えるようにした。

このルビを振るためのライブラリは"satysfi-ruby"として GitHub で公開・配布した。[10] また、私の運営しているブログにも SATYSFI でルビ振り機能を実装した話を記事として公開した。[11]

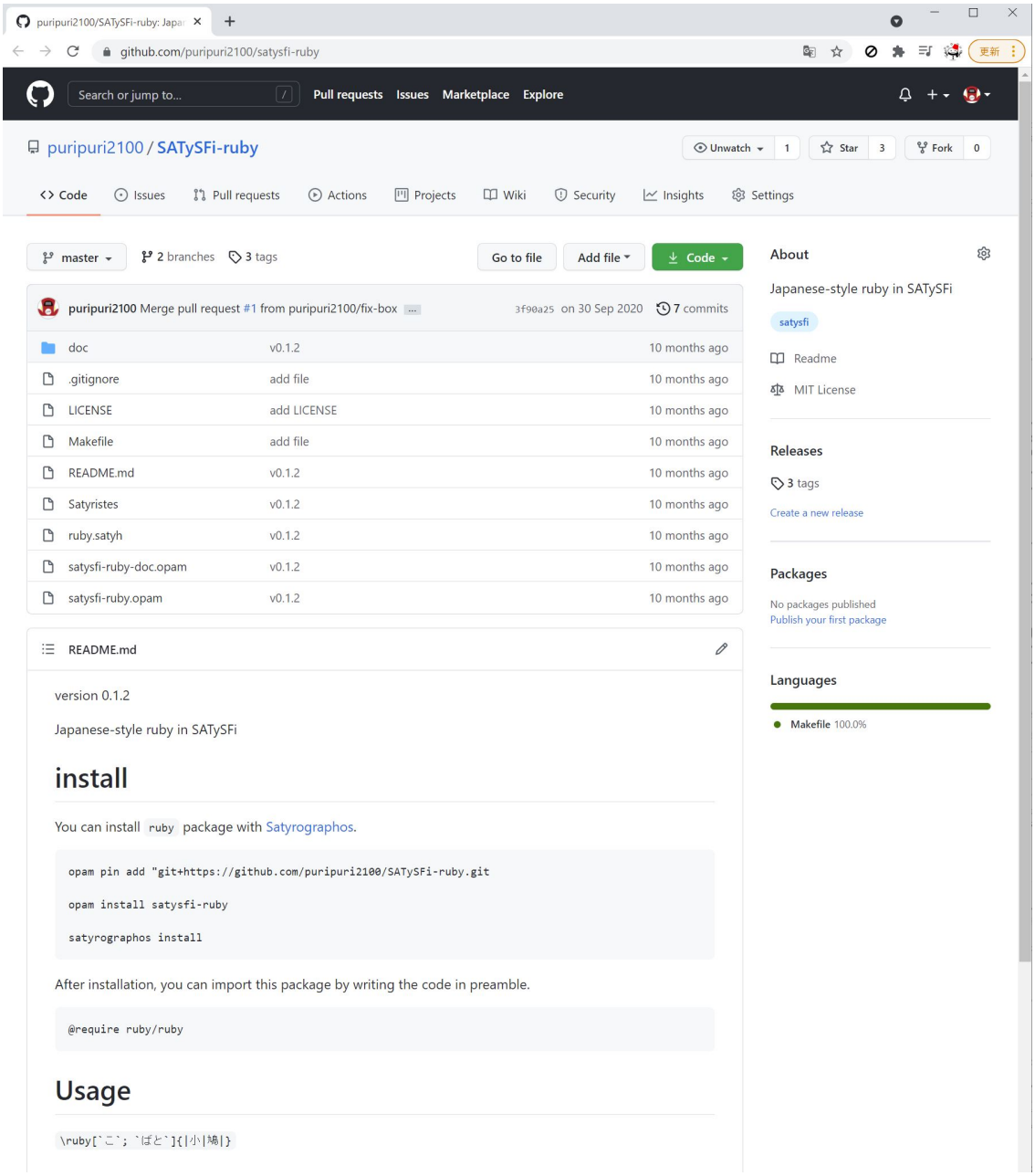


図 11 satysfi-ruby のリポジトリ

1.3.2. satysfi-json の作成

satysfi-json は SATySFi で実装された JSON と JSON5 のパーサとデコーダである。

JSON とはデータ形式フォーマットの一つである。Web アプリの作成でデータの受け渡しをする際によく使われるほか、ソフトウェアの設定などにも使われる。SATySFi でもフォントファイルのデータや相互参照のためのダンプファイルの形式として採用されている。とてもメジャーで一般的なものであり、この世界で実際に利用されているプログラミング言語のほとんどで JSON 形式をパースするライブラリが実装されていると言っても過言ではない。JSON の規格はjson.orgに書かれている。

現在 SATySFi では相互参照には文字列しか埋め込むことができない。そのため、数字

や真偽値、データを埋め込むことや逆にそのようなデータを取り出すことができない。真偽値を埋め込み、取り出したいが、これができないために困ったことがあった。そこで、数字や真偽値を埋め込む機能を SATySFi 本体に導入することを提案したことがあった。しかし、SATySFi の作者から"It seems to me that they should be implemented as ordinary user- or package-level functions, not provided as primitives; it would be simpler to make users or packages responsible for encoding/decoding values (by using satysfi-base, for example) than to add somewhat duplicated primitives."[12] と返答があり、この機能の実装を断念した。結局この問題はライブラリ上で数字や真偽値のパーサを実装することで解決したが、SATySFi で JSON パーサとデコーダを実装していればより汎用的なデータを取り扱うことができることに気が付いた。そこで、これを実装した。

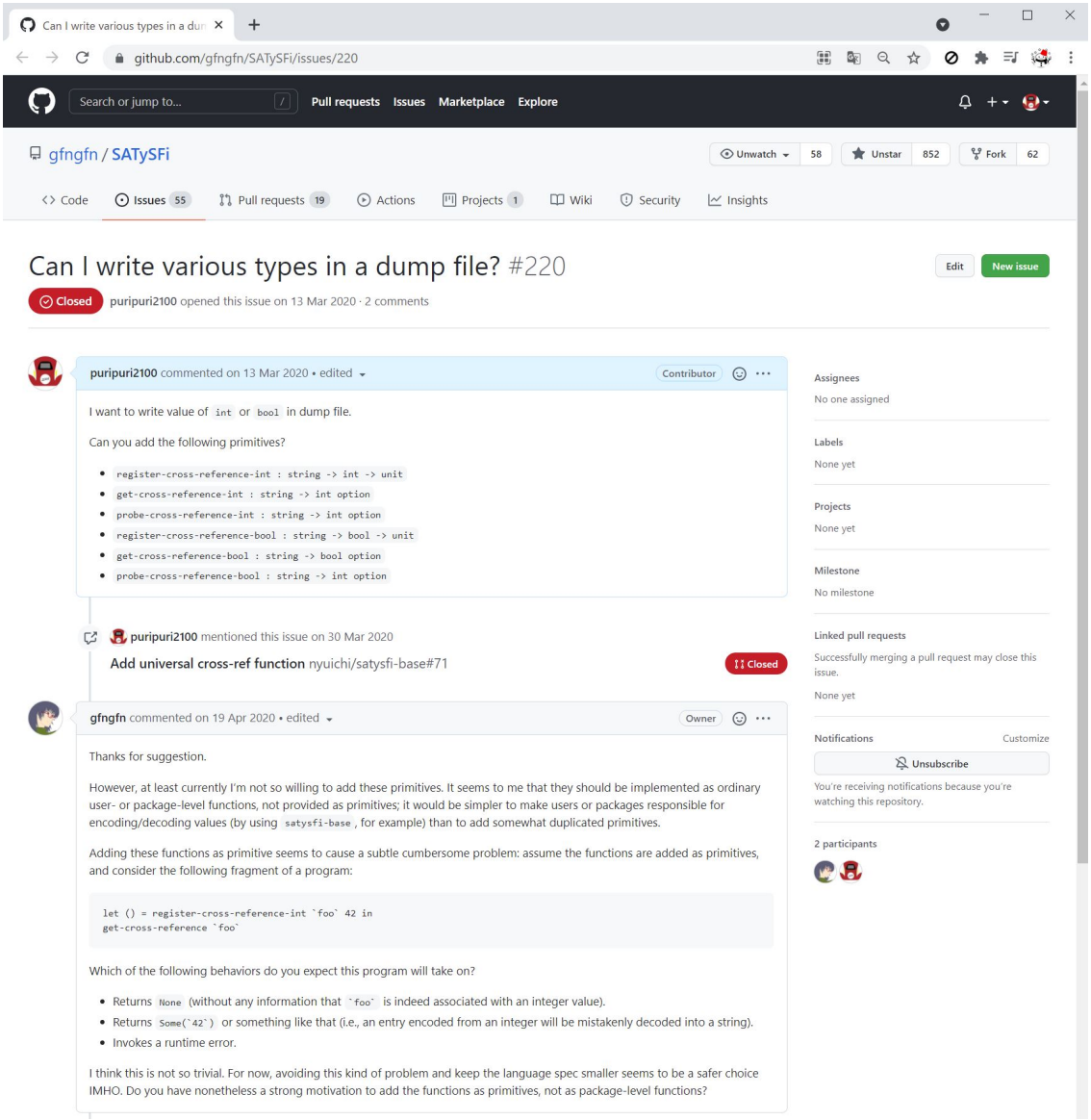


図 12 issue#220 のスクリーンショット [12]

SATySFi では文字列を一文字ずつ分解して扱う方法があることと、JSON の文字列と数字の表現が複雑なこと、そして JSON の文法が左再帰となっておらず、左再帰下降パーサ

で処理できることから、手書き lexer によって JSON 文字列をトークン化し、そのトークンリストを手書きの左再帰下降パーサで解析して SATYSFI のデータ構造に落とし込むという方法を採用した。

lexer は文字列からトークン列を生成する関数である。文字列を頭から一文字ずつ読みこんで変換していく。読み込んだ文字によっては複数文字先読みして複数の文字列にしてから変換することもある。トークンは

- Comma（","のこと）
- Colon（":"のこと）
- LeftCurlyBracket（"{"のこと）
- RightCurlyBracket（"}"のこと）
- LeftSquareBracket（"["のこと）
- RightSquareBracket（"]"のこと）
- Null（"null"のこと）
- True（"true"のこと）
- False（"false"のこと）
- Int of int（整数を表す）
- Float of float（小数を表す）
- String of string（文字列を表す）

の 12 個用意した。

lexer は再帰関数で実装した。引数は「トークンのリスト」・「エラーメッセージ使用する文字の位置情報」・「文字のリスト」の 3 つである。終了条件は文字のリストが空になるまで、である。

空白文字は無視するように規定されているため、規格の上で空白文字とされている"space"・"Horizontal Tab"・"Carriage Return"・"LineFeed"の 4 種類の文字のいずれかを読み込んだときには、その文字だけ消費し、文字位置を 1 つ進めるだけとした。

一文字をあらわすトークン（Comma や LeftCurlyBracket、RightSquareBracket などである）については、該当する文字が現れた時には、対応するトークンをトークンのリストに追加し、その文字を消費し、文字位置も一つ進める。

null・ true・ false の 3 つについては、まず、現れた文字がアルファベットであった場合に、次の文字がアルファベットでは無くなるまで文字の取得を行い続ける。次に、取得した文字をつなげ、それらが null・ true・ false のいずれかであった場合には該当するトークンへと変換し、トークンのリストに追加する。3 つのいずれでも無かった場合はエラーとする。最後に文字位置と文字のリストをアルファベット文字列を取得した後のものに更新し、lexer の再帰関数に渡すようにした。

JSON での文字列の表記法は基本的に単純ではあるものの、エスケープされた文字列の扱いがやや面倒であった。

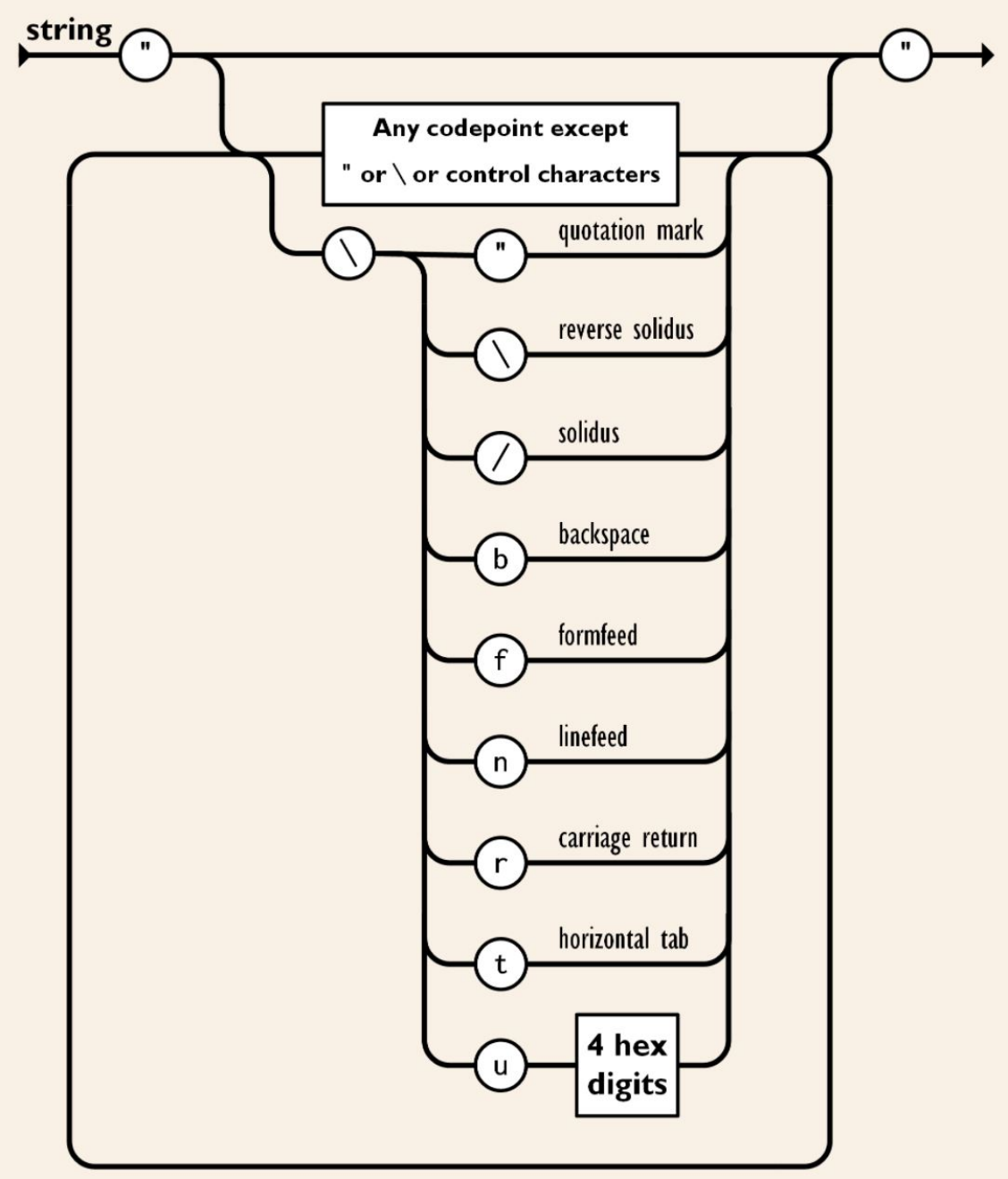


図 13 文字列の構文を表した図 [13]

文字列はダブルクオーテーション (") から始まり、ダブルクオーテーションで終わる。そのため、読み込んだ文字がダブルクオーテーションであった場合は、次にダブルクオーテーションが現れるまで、出てきた文字を全て stack に保存し、ダブルクオーテーションが現れた時に、保存していた文字列を String トークンとするという実装をした。ただし、バックスラッシュ (\) が現れた場合は、文字列のエスケープモードに入るため、特殊な実装を必要とする。上の図にあるように、バックスラッシュの後に

- "
- \
- /
- b

- f
- n
- r
- t

が現れた場合は、それに対応する文字や特殊文字に置き換えて stack に保存しなければならない。このとき、文字列位置は二つ移動させ、文字のリストも当然 2 つ消費する。また、バックスラッシュのあとに u があり、さらにその後に「0 から 9 までの数字、もしくは A から F、もしくは a から f のアルファベット」が 4 つ並んでいた場合は、その 4 つの文字を 16 進数表記と見た時の数字を Unicode ポイントとして持っている文字を stack に保存する。この時は文字位置を 6 つ進め、文字を 6 つ消費することになる。もし、\u の後に「0 から 9 までの数字、もしくは A から F、もしくは a から f のアルファベット」が 4 つ並んでいなかった場合[†1]はエラーを返すようにする。

整数と小数は JSON 規格の上では number とされ、同じものとして扱われている。しかし、SATYSF$_{\mathrm{I}}$ は整数と小数を型レベルで分けているため、satysfi-json は SATYSF$_{\mathrm{I}}$ の設計に従い、整数と小数に分けてデータを取得する。数字の表記は下の図14のように定められている。

1：例えば 3 つしか並んでおらず、4 つ目には"X"という文字が来ていた場合など

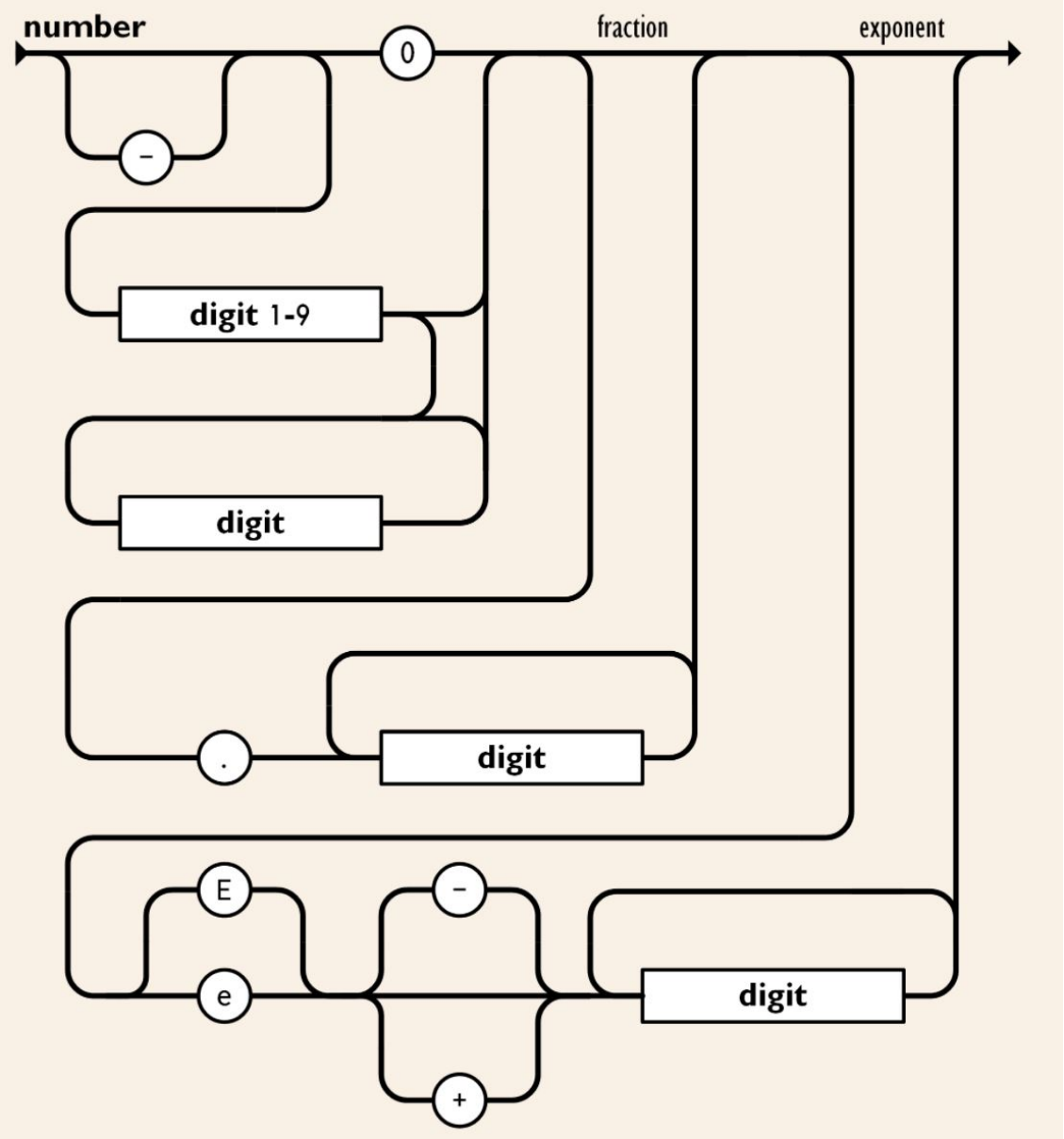


図 14 数字表現の構文を表した図 [13]

数字かマイナス記号のどちらかが現れたときに number の解析に入る。まず数字が続く限り文字を取得し続ける。その後、次の文字が小数点かどうかで分岐が発生する。小数点があった場合は、その後に続く数字を取得する。その後に指数表記があるかどうかをチェックし、指数表記があった場合にはその部分も取得する。そして数字の列と小数点を結合してパースし、小数に変換する。指数表記があった場合にはその計算も行う。小数点がなかった場合は指数表記のチェックに入り、あった場合には同様に取得する。そして文字列をパースして整数に変換した後に、指数部分の計算を行う。

これで全ての表記規則をトークンに変換することができた。次は、こうして出来上がった lexer を通して作成したトークン列をパースし、$\mathrm{SATySF_I}$ 上のデータ構造に変換する。$\mathrm{SATySF_I}$ でのデータ構造は

```
type json =
```

```
  | Null
  | Bool of bool
  | Int of int
  | Float of float
  | String of string
  | Array of json list
  | Object of (string * json) list
```

で定義される再帰型である。

パーサは 1 つ先読みの再帰下降構文解析法（LL(1) 法とも呼ばれる）で実装する。この解析方法は、トークン列を左側から 1 つずつ読んでいくことで解析でき、バックトラックなどを行わないで効率的に解析できる方法として知られている。

SATySF$_{\mathrm{I}}$ は相互再帰関数を定義することができるため、今回作成する LL(1) パーサの実装が容易であった。

まず、value と JSON の規格で呼ばれているものをパースする関数を作成した。

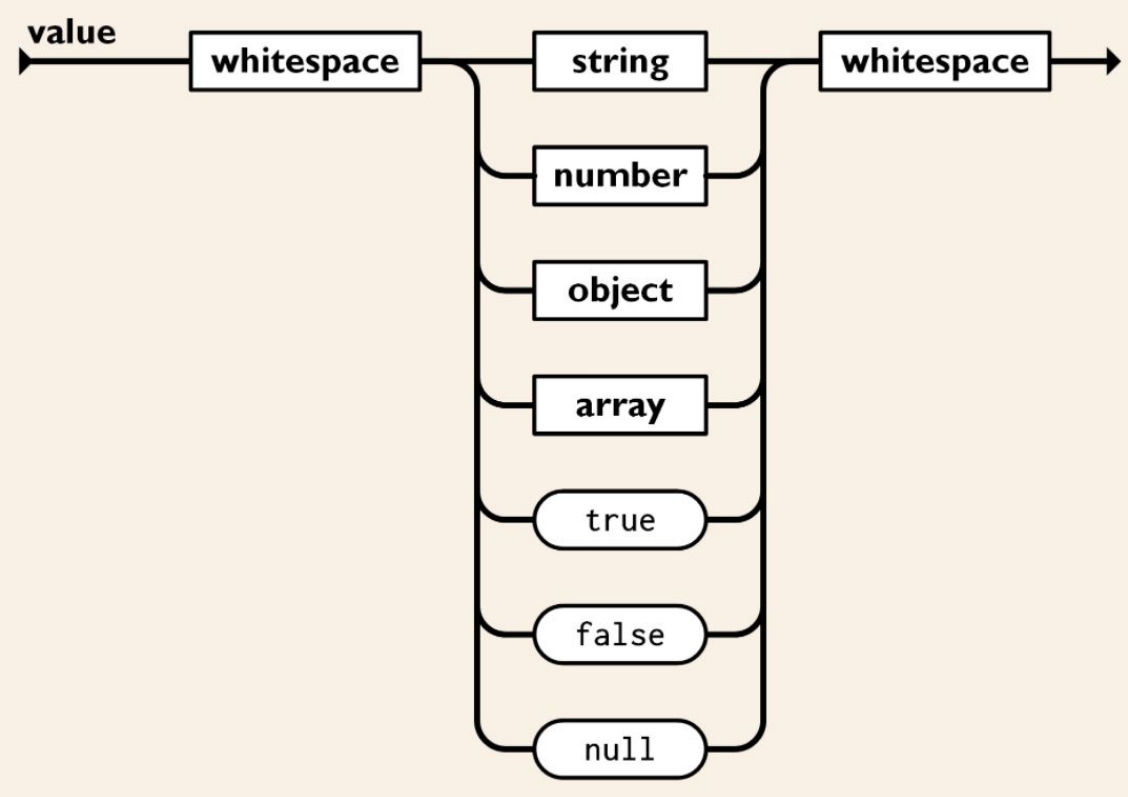


図 15 value の構文を表した図 [13]

ここではトークンとしての

- String
- Int
- Float

- True
- False
- Null

が現れた時に、それに対応するデータ構造に中身を移し、データ構造列に加える処理を行う。また、object と array の解析のために、トークン LeftSquareBracket と LeftCurlyBracket のどちらかが現れた時に、相互再帰関数として同時に定義している「array を解析する再帰関数」と「object を解析する再帰関数」をそれぞれ呼び出すようにした。

array の定義は下の図のようになっている。

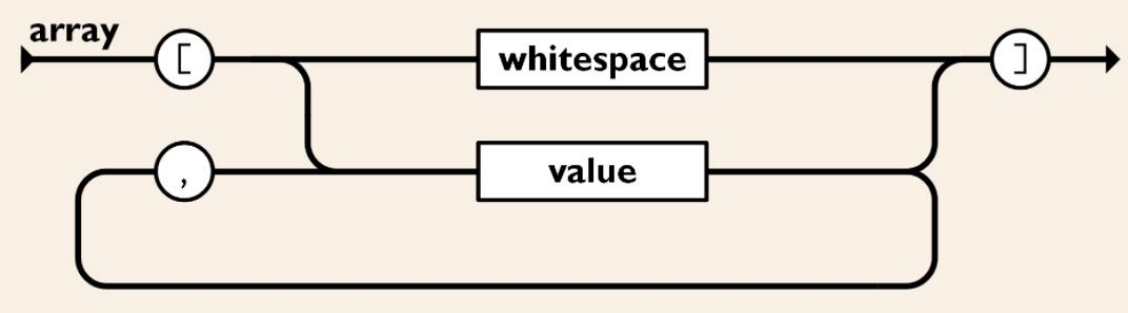


図 16 配列の構文を表した図 [13]

array を解析する関数には、"["が既に消費された時点のトークンのリストが渡される。また、whitespace はトークンに変換する時点で取り除かれているため、考えなくて良い。まず、「value を解析する再帰関数」を呼び出し、value として扱われるデータと、value を解析した後に残るトークンのリストの 2 つのデータを作成する。残ったトークン列の先頭が"]"であった時には array が終了したと判定し、関数の処理を終了させる。残ったトークン列の先頭がカンマであった場合には array が続くと判定し、カンマの次のトークンに対して再度「value を解析する再帰関数」を適用させ、データを取得する。これを終了するまで繰り返す。

object の定義は下の図のようになっている。

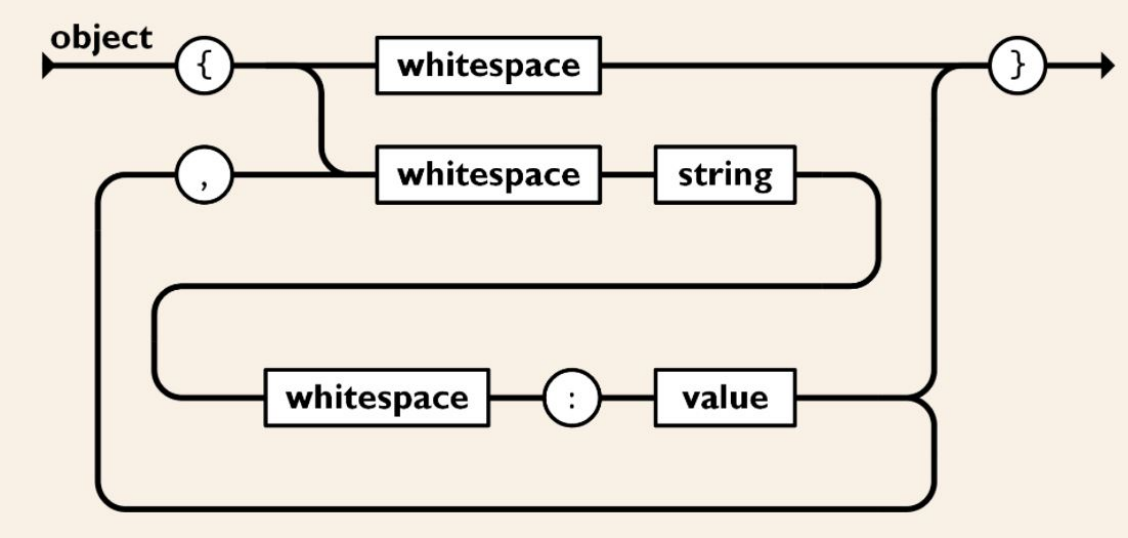


図 17 object の構文を表した図 [13]

これも array のときと同様に、"{"が既に消費された時点でのトークン列が与えられ、whitespace は全て除去されている。array と同様に処理していくが、array の定義が value 単体であったのとは異なり、string とコロンと value がセットとなっている。これらを順に解析してデータ化する。そして"}"が現れるまで、コロンとセットで解析・消費していく。

以上のように定義した「value を解析する再帰関数」・「array を解析する再帰関数」・「object を解析する再帰関数」について、最初に「value を解析する再帰関数」を呼び出すと、トークンに応じてそれぞれがお互いを呼び出しあい、最終的にトークンが全て消費されて処理が綺麗に終われば解析成功となる。もし途中でトークンが尽きたり、逆に解析が終了したのにもかかわらずトークンが余っているような時には解析が失敗したとしてエラーを返すようにしている。また、予期せぬトークンが現れた時もエラーを返して終了するように定義してある。

これで JSON のパーサの実装は終わったが、現在のところ純粋な JSON は制限が多くて使いにくいという評価が世界ではなされている。そのため、その制限を緩和し、機能を拡張させた"JSON5"と呼ばれる規格が流行っている。そのため、今回は JSON のパーサだけではなく、JSON5 のパーサも実装することとした。JSON5 の公式ページはhttps://json5.org/である。ここに「どのような拡張が JSON に対してなされているのか」という説明が書かれている。例えば、

- コメントを使用することができる
- array や object での末尾カンマを許可する
- 文字列は一重引用符で囲んで表現しても良い
- 改行文字をエスケープすることで長い文字列を複数行で表すことができる
- 数字で正負の無限大と NaN を使うことができる
- 数字をプラス記号で始めることができる
- オブジェクトのキーが英数字である場合は文字列を使わなくて良い

といったものである。コメント・文字列・数字表記部分についての JSON 規格との差異については lexer の改修で対応した。そして末尾カンマの許容やオブジェクトのキーについてはパーサに手を加えて対応した。

パーサを作成し終えたため、今度は SATγSF$_{\mathrm{I}}$ で JSON・JSON5 を表すデータ構造を JSON 形式・JSON5 形式の文字列に変換するデコーダを実装した。パーサとデコーダが揃うことで、データ形式の入出力が容易になる。文字列のエスケープが難しいだけで、他は再帰的に処理していけば文字列への変換が出来る。

作成した JSON パーサがおかしな挙動をしていないかのテストも行った。"Parsing JSON is a Minefield 💣"[14] という、JSON パーサのコーナーケースとテストに関するブログを参考にし、そのテストケースが置いてあるリポジトリ [15] の中身を基にテストを作成した。

その結果、「エラーを出さなければならないにもかかわらず受理してしまったケース」が全 188 件中 10 件、逆に「受理しなければならないのにもかかわらずエラーを出してしまったケース」が全 95 件中 6 件出た。これはとても少ないと言える。「エラーを出さなければならないにもかかわらず受理してしまったケース」の全てが数字の処理と文字列の処理で失敗していた。「受理しなければならないのにもかかわらずエラーを出してしまったケース」の全てが Unicode ポイントで表記された文字の処理での失敗であった。このテストケースはhttps://github.com/puripuri2100/SATySFi-json-testに置いてある。

このパーサーは"SATySFi-json"という名前をつけ、GitHub のリポジトリで公開している。[16]

2. 学校活動でのソフトウェア等の作成

インターネット上に出すのが良く無さそうなので削除

おわりに

あとがきです。

参考文献

[1] 開成学園 . "ペン剣基金助成研究公募 *2020*". https://kaiseigakuen.jp/wp/wp-content/uploads/2020/06/2020_penken.pdf, 2020. (accessed 2021-7-28).

[2] Naoki Kaneko, T. Suwa, and GitHub. "*A class file easy to customize with SATySFi*". https://github.com/puripuri2100/exdesign. (accessed 2021-7-28).

[3] T. Suwa and GitHub. "*add string-explode #202*". https://github.com/gfngfn/SATySFi/pull/202, 2019. (accessed 2021-7-28).

[4] T. Suwa and GitHub. "*Add string cmpare functions #127*". https://github.com/nyuichi/satysfi-base/pull/127, 2020. (accessed 2021-7-28).

[5] Naoki Kaneko and GitHub. "*SATySFi packages that convert numbers to the following notation.*". https://github.com/puripuri2100/SATySFi-num-conversion. (accessed 2021-7-28).

[6] T. Suwa and GitHub. "*Add read external file support #200*". https://github.com/gfngfn/SATySFi/pull/200, 2021. (accessed 2021-7-28).

[7] T. Suwa and GitHub. "*I want to be able to set up a document information dictionary #267*". https://github.com/gfngfn/SATySFi/issues/267, 2021. (accessed 2021-7-28).

[8] T. Suwa and GitHub. "*set up a document information dictionary #268*". https://github.com/gfngfn/SATySFi/pull/268, 2021. (accessed 2021-7-28).

[9] W3C. "*Ruby and Emphasis Dots* ルビと圏点処理". https://www.w3.org/TR/jlreq/#ruby_and_emphasis_dots, 2020. (accessed 2021-7-28).

[10] Naoki Kaneko and GitHub. "*Japanese-style ruby in SATySFi*". https://github.com/puripuri2100/SATySFi-ruby. (accessed 2021-7-28).

[11] puripuri2100 and Hatena Blog. "*SATySFi* でルビを使う〜 *SATySFi-ruby* パッ

ケージ〜". https://puripuri2100.hatenablog.com/entry/2020/12/01/202913, 2020. (accessed 2021-7-28).

[12] T. Suwa and GitHub. "*Can I write various types in a dump file? #220*". https://github.com/gfngfn/SATySFi/issues/220, 2021. (accessed 2021-7-28).

[13] json.org. "*Introducing JSON*". https://www.json.org/json-en.html. (accessed 2021-7-28).

[14] Nicolas Seriot. "*Parsing JSON is a Minefield* 💣". http://seriot.ch/parsing_json.php, 2016. (accessed 2021-7-28).

[15] Nicolas Seriot and GitHub. "*A comprehensive test suite for RFC 8259 compliant JSON parsers*". https://github.com/nst/JSONTestSuite, 2016. (accessed 2021-7-28).

[16] Naoki Kaneko and GitHub. "*json parser for SATySFi*". https://github.com/puripuri2100/SATySFi-json. (accessed 2021-7-28).

[17] puripuri2100 and Hatena Blog. "*TeXConf 2019* の供養". https://puripuri2100.hatenablog.com/entry/2019/12/22/151426, 2019. (accessed 2021-7-28).

[18] TeXConf 2019. "*Second Announcement*（参加登録受け付け）". https://texconf2019.tumblr.com/post/186806708591/second-announcement, 2019. (accessed 2021-7-28).

[19] Naoki Kaneko and GitHub. "*Convert Markdown to LaTeX with SATySFi.*". https://github.com/puripuri2100/satysfi-md2latex. (accessed 2021-7-28).

[20] connpass. "*Online.tex 2020*". https://connpass.com/event/188075/. (accessed 2021-7-28).

[21] 金子尚樹 . "*SATySFi* を使用した *Markdown* から *LaTeX* へのファイル変換について". https://speakerdeck.com/puripuri2100/satysfiwoshi-yong-sitamarkdownkaralatexhefalsehuairubian-huan-nituite, 2020. (accessed 2021-7-28).

[22] Naoki Kaneko and GitHub. "*SATySFi font package for Noto Emoji fonts.*". https://github.com/puripuri2100/SATySFi-fonts-noto-emoji. (accessed 2021-7-28).

[23] @kakinaguru_zo and Qiita. "*LuaTeX* でカラーフォント・絵文字 🐰 *(SVG/*

COLR/CBDT)". https://qiita.com/kakinaguru_zo/items/8a085484c7a0530e032b, 2020. (accessed 2021-7-28).

[24] Naoki Kaneko and GitHub. "*Convert SVG file to SATySFi's file.*". https://github.com/puripuri2100/svg2saty. (accessed 2021-7-28).

[25] puripuri2100 and Hatena Blog. "絵文字結合を実装する". https://puripuri2100.hatenablog.com/entry/2020/05/06/231925, 2020. (accessed 2021-7-28).

[26] twitter and GitHub. "*Emoji for everyone. https://twemoji.twitter.com/*". https://github.com/twitter/twemoji. (accessed 2021-7-28).

[27] Naoki Kaneko and GitHub. "*[WIP] To use color emoji with SATySFi*". https://github.com/puripuri2100/satysfi-twemoji. (accessed 2021-7-28).

[28] Naoki Kaneko and GitHub. "*A mdbook backend for generating SATySFi documents.*". https://github.com/puripuri2100/mdbook-satysfi. (accessed 2021-7-28).

[29] The Rust Programming Language and GitHub. "*Create book from markdown files. Like Gitbook but implemented in Rust*". https://github.com/rust-lang/mdBook. (accessed 2021-7-28).

[30] Steve Klabnik, Carol Nichols, Contributions from the Rust Community, and GitHub. "*The Rust Programming Language*". https://doc.rust-lang.org/book/. (accessed 2021-7-28).

[31] The Rust Programming Language and GitHub. "*The Rust Programming Language*". https://github.com/rust-lang/book. (accessed 2021-7-28).

[32] The Rust Community. "*Rust By Example*". https://doc.rust-lang.org/stable/rust-by-example/. (accessed 2021-7-28).

[33] Alex Crichton, Steve Klabnik, Carol Nichols, and Contributions from the Rust Community. "*The Cargo Book*". https://doc.rust-lang.org/cargo/. (accessed 2021-7-28).

[34] Rust Language Community. "*Rust Cookbook*". https://rust-lang-nursery.github.io/rust-cookbook/. (accessed 2021-7-28).

[35] Michael Bryan and GitHub. "*An experimental mdbook backend for*

creating EPUB documents.".
https://github.com/Michael-F-Bryan/mdbook-epub. (accessed 2021-7-28).

[36] Liam Beckman and GitHub. "*An mdbook backend for generating LaTeX and PDF documents.*".
https://github.com/lbeckman314/mdbook-latex. (accessed 2021-7-28).

[37] 諏訪敬之 . "静的型つき組版処理システム *SATySFi @* 第 *61* 回プログラミング・シンポジウム". https://www.slideshare.net/bd_gfngfn/satysfi-61. (accessed 2021-7-28).

[38] The Tectonic Project and GitHub. "*A modernized, complete, self-contained TeX/LaTeX engine, powered by XeTeX and TeXLive.*".
https://github.com/tectonic-typesetting/tectonic. (accessed 2021-7-28).

[39] Raph Levien and GitHub. "*This library is a pull parser for CommonMark, written in Rust.*".
https://github.com/raphlinus/pulldown-cmark. (accessed 2021-7-28).

[40] Mathias and GitHub. "*A simple and general purpose html/xhtml parser, using Pest.*".
https://github.com/mathiversen/html-parser. (accessed 2021-7-28).

[41] Naoki Kaneko and GitHub. "*This software converts xml file to SATySFi's document file.*".
https://github.com/puripuri2100/xml2saty-rust. (accessed 2021-7-28).

[42] commonmark.org. "*CommonMark - A strongly defined, highly compatible specification of Markdown -*".
https://commonmark.org/. (accessed 2021-7-28).

[43] The Rust Programming Language. "*mdBook-specific features*".
https://rust-lang.github.io/mdBook/format/mdbook.html. (accessed 2021-7-28).

[44] The Rust Programming Language. "*Alternative Backends*".
https://rust-lang.github.io/mdBook/for_developers/backends.html. (accessed 2021-7-28).

[45] Naoki Kaneko and GitHub. "*A SATySFi class file for processing document file generated by mdbook-satysfi*". https://github.com/puripuri2100/

satysfi-class-mdbook-satysfi. (accessed 2021-7-28).

[46] monaqa and Twitter. "ツイート *@monaqa*". https://twitter.com/mo_naqa/status/1362944222433734660. (accessed 2021-7-28).

[47] cluster. "*SATySFi Conf 2020*". https://cluster.mu/e/9a567d0b-a7ab-4758-8c0f-16236b36dffd. (accessed 2021-7-28).

[48] connpass. "*SATySFi Conf 2020*". https://connpass.com/event/174844/. (accessed 2021-7-28).

[49] connpass. "*SATySFi Conf 2021*". https://connpass.com/event/206277/. (accessed 2021-7-28).

[50] 金子尚樹 . "*mdbook-satysfi* を作成しました". https://speakerdeck.com/puripuri2100/mdbook-satysfiwozuo-cheng-simasita, 2021. (accessed 2021-7-28).

[51] @puripuri2100. "*mdbook* のプラグインを作る話". https://speakerdeck.com/puripuri2100/mdbookfalsepuraguinwozuo-ruhua, 2021. (accessed 2021-7-28).

[52] Naoki Kaneko and GitHub. "*Make LL(1) token parser code for Rust*". https://github.com/puripuri2100/llmaker. (accessed 2021-7-28).

[53] κeen, 河野達也 , and 小松礼人 . "実践 *Rust* 入門［言語仕様から開発手法まで］". 技術評論社 , 2019.

[54] puripuri2100 and Hatena Blog. "*Rust* 用の *LL(1)* トークンパーサジェネレータを作った話". https://puripuri2100.hatenablog.com/entry/2020/06/21/211654, 2020. (accessed 2021-7-28).

[55] YouTube. "*SATySFi Conf 2020*". https://www.youtube.com/playlist?list=PL0VFdv9CLkQ3nAE46ZvEztjQauzH0haZb, 2020. (accessed 2021-7-28).

[56] zptmtr and Twitter. "ツイート *@zptmtr*". https://twitter.com/zptmtr/status/1286950873088659458, 2020. (accessed 2021-7-28).

[57] Naoki Kaneko and GitHub. "電気係貸出管理ソフトウェア". https://github.com/puripuri2100/dlm. (accessed 2021-7-28).

[58] 開成学園パズル研究部 . "開成学園パズル研究部 *HP*". https://nipori.gitlab.io/nipori. (accessed 2021-7-28).

[59] 開成学園パズル研究部 . "ニポリ *16* 号". https://nipori.gitlab.io/nipori/nipori16/html/nipori16.html. (accessed 2021-7-28).